DOM

Desarrollo web en entorno cliente

José María Torresano
Noviembre 2018
daw.cierva@gmail.com

1. Introducción

2. Trabajando con el DOM

3. Localizando elementos

4. Modificando el DOM

1. INTRODUCCIÓN

DOM o Document Object Model es un conjunto de utilidades específicamente diseñadas para manipular documentos XML, XHTML y HTML.

Técnicamente, DOM es una API de funciones que se pueden utilizar para manipular las páginas HTML de forma rápida y eficiente desde cualquier lenguaje que soporte la API.

Antes de poder utilizar sus funciones, el navegador transforma internamente el archivo HTML original en una estructura más fácil de manejar formada por una jerarquía de nodos interconectados en forma de árbol (DOM).

El árbol generado no sólo representa los contenidos del archivo original (mediante los nodos del árbol) sino que también representa sus relaciones (mediante las ramas del árbol que conectan los nodos).

El navegador

1 El navegador carga la página **HTML** que recibe del servidor

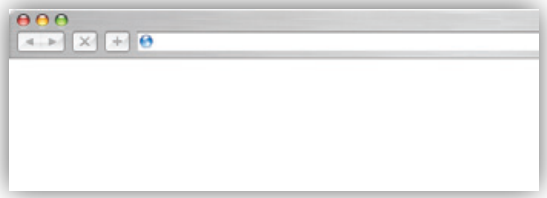
index.html

2 El motor del navegador construye un '**document**' a partir del archivo HTML y CSS cuyo resultado es el DOM

index.html

La versión DOM de la página

3 El navegador muestra la página en su **viewport**



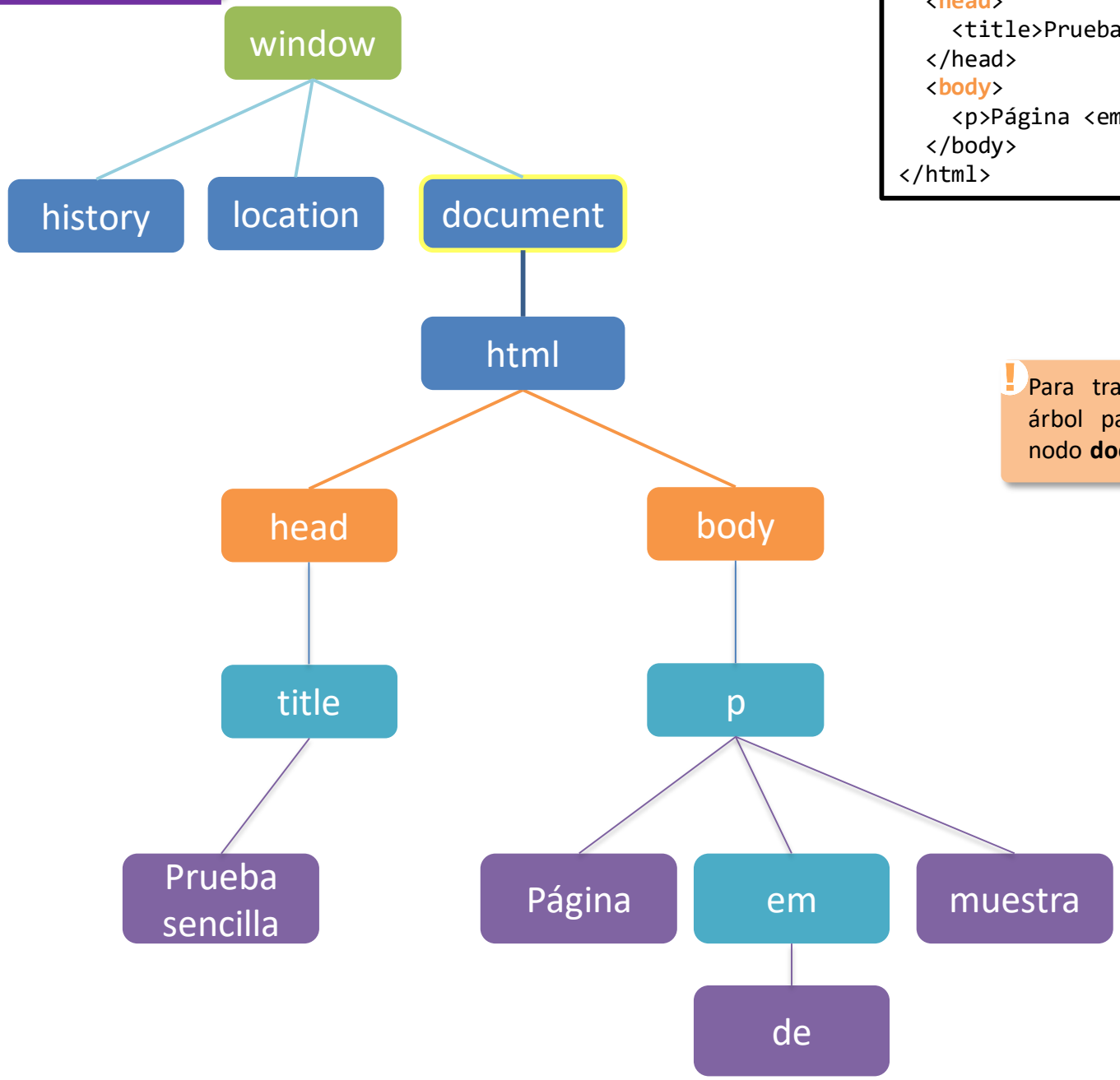
4 El interprete de JS utiliza el DOM para cambiar la página sin tener que recargarla

Interprete JavaScript

La versión DOM de la página

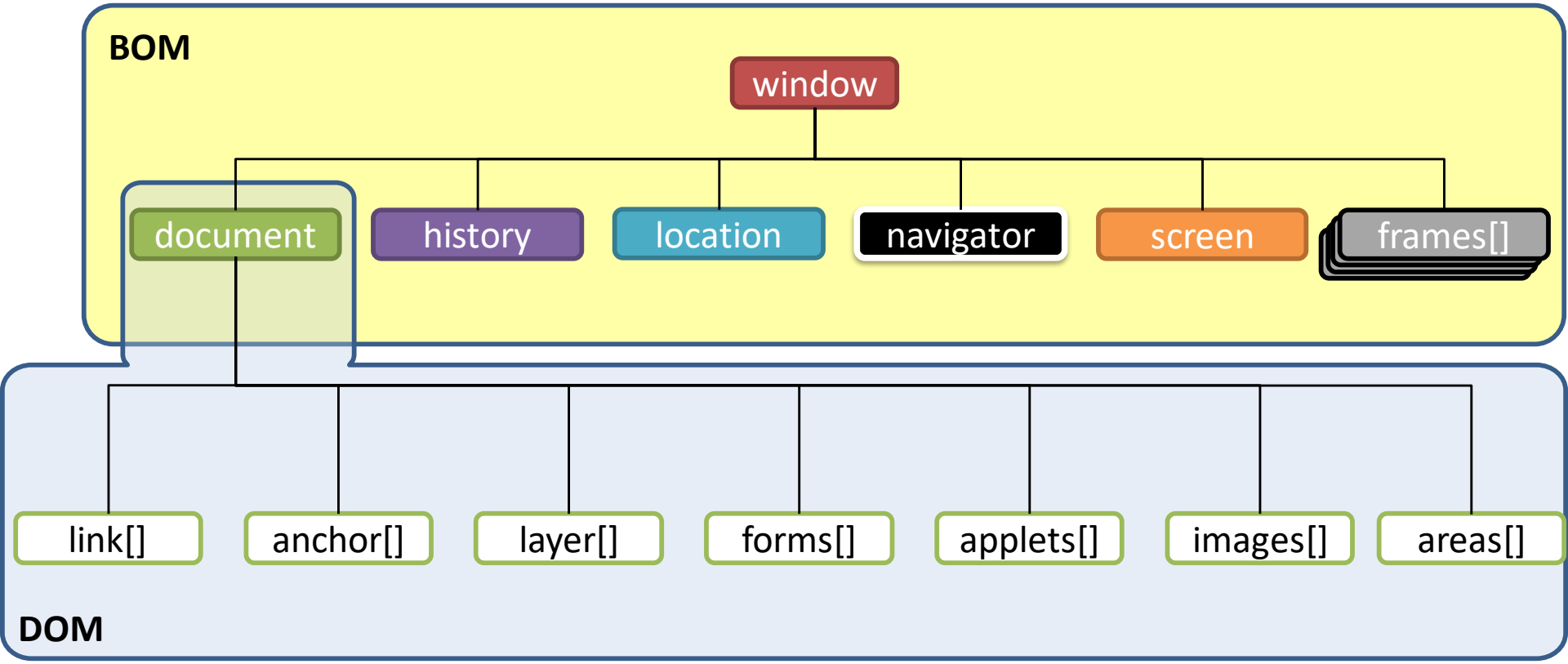
{cliente} daw.cierva@gmail.com

```
<html>
  <head>
    <title>Prueba sencilla</title>
  </head>
  <body>
    <p>Página <em>de</em> muestra</p>
  </body>
</html>
```



! Para trabajar con el árbol partiremos del nodo **document**

DOM1.0


BOM
window
document
history
location
navigator
screen
frames[]
link[]
anchor[]
layer[]
forms[]
applets[]
images[]
areas[]
DOM

DOM1.0

Colecciones HTML del DOM

Nombre	Descripción
`document.all`	Todos los elementos de la pagina. Solo en IE 4+.
`document.forms`	Todos los formularios.
`document.style`	Los objetos de las hojas de estilos enlazadas por el archivo o incluidos en la misma se encuentren donde se encuentren
`document.images`	Todas las imágenes.
`document.applets`	Todas las apples de Java.
`document.plugins`	Todos los nodos < embed > de la pagina.
`document.embeds`	Otra referencia a los nodos < embed > y < object > de la pagina.
`document.links`	Todos los enlaces de la pagina (< a >).

Se recomienda NO utilizar las colecciones para localizar un elemento

2. TRABAJANDO CON EL DOM

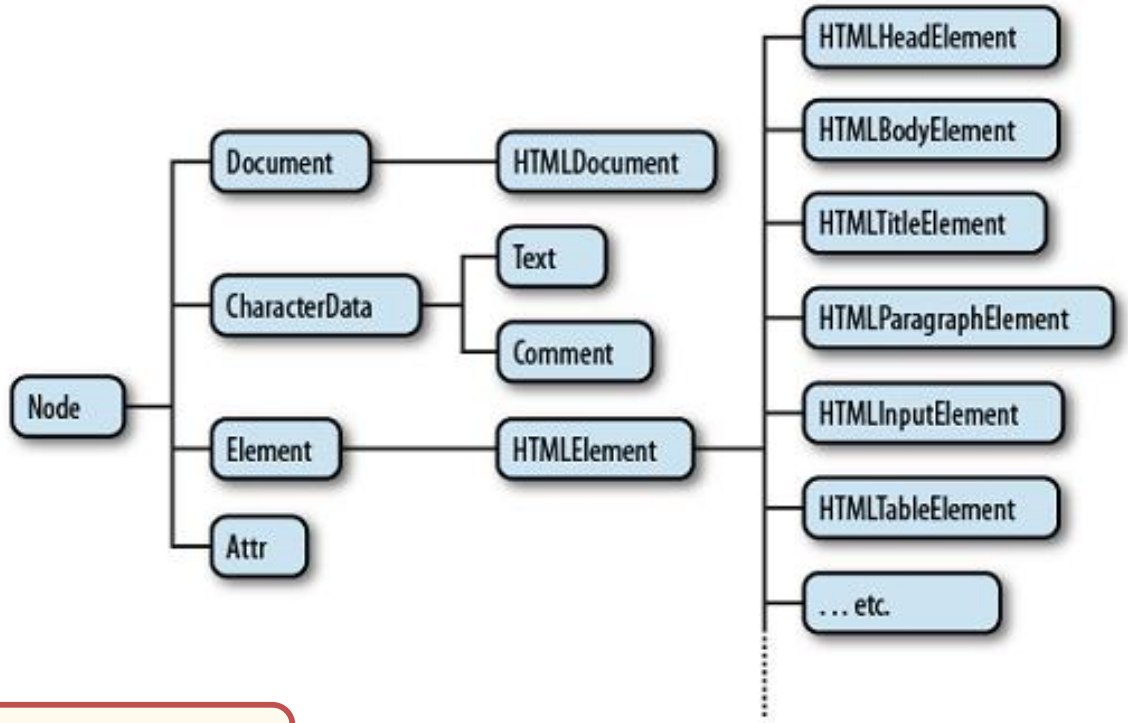
Objetos del tipo Node

Todo nodo en el DOM tiene asociado un tipo, identificado por un valor numérico, guardado en la propiedad `nodeType`.

Tipo		Valor
Elementos HTML	`<body> <a> <p> <h1> <html>`	1
Atributos de elementos HTML	`class="unaClase"`	2
Texto	Caracteres de texto incluidos retornos de carro y espacios	3
Comentario	Comentarios HTML	8
El documento (`document`)	`<!DOCTYPE html>`	9

Objetos del tipo Node

Cada nodo hereda las propiedades y métodos de Node. Dependiendo del tipo de nodo hay subnodos que extienden el objeto Node.



Los Node son objetos del DOM no de JS

{cliente} daw.cierva@gmail.com

Array vs NodeList vs HTMLCollection

Por lo general, Firefox y IExplorer devolvían `HTMLCollection`; Chrome, Safari, Firefox y Opera devolvían `NodeList`.

A día de hoy (noviembre de 2018), dependiendo del método utilizado, devuelven `NodeList` o `HTMLCollection`.

`NodeList` es una colección de nodes. `HTMLCollection` es una colección de elementos HTML/XML. `NodeList` es un contenedor más general. Por ejemplo, el texto que contiene un elemento <p> es un `Node` no un elemento HTML.

`NodeList` y `HTMLCollection` son objetos parecidos a los array de JS, pero no lo son. Aún así `NodeList`, en algunas implementaciones, cuenta con el método `forEach`. Ambos cuentan con la propiedad `length` y se pueden recorrer mediante índices. No se puede utilizar la estructura *`for … in`* ni los métodos de array de JS.

Array vs NodeList vs HTMLCollection

Los elementos que componen las `NodeList` y `HTMLCollection` pueden ser referencias **fijas** (o estáticas) o referencias **volátiles** (o vivas) a los nodos del DOM. En el primer caso, el agregar o eliminar un nodo al DOM, después de haber hecho la selección, si la cumple, no se refleja en la colección o lista; en el segundo caso, sí se reflejará sin necesidad de volver a hacer la selección. **En ambos casos, los datos que contienen los nodos están vivos**.

Cuando se vean, más adelante, los métodos de selección se indicará si la colección o la lista está *viva* o no.

{cliente} daw.cierva@gmail.com

NodeList

La interfaz `NodeList` representa una colección ordenada de `Nodes`, indexados por número (a partir de cero).

Una `NodeList` puede ser una colección ***viva (o dinámica)*** o ***estática.***

Los métodos `getElementsByName` y `querySelectorAll` devuelven este tipo de objeto. El primer método lo devuelve *dinámico*; el último, *estático.*

```
var nodesArray = Array.prototype.slice.call(nodeList);
o var nodesArray = [].slice.call(nodeList);
o var nodesArray = Array.from(nodeList); [ES6]
```

Como convertir un NodeList en un Array de JS para utilizar los métodos de los Arrays

En alguna navegadores puede existir el método forEach equivalente al de los arrays de JS

HTMLCollection

La interfaz `HTMLCollection` representa una colección ordenada de `Elements`, indexados por número (a partir de cero).

Igual que una `NodeList`, una `HTMLCollection` puede ser una colección ***viva*** o una colección ***estática***.

Según la W3C (DOM4): *Elements is the better solution for representing a collection of elements. HTMLCollection is an historical artifact we cannot rid the web of.*

```
var elementsArray = Array.prototype.slice.call(HTMLCollection);
var elementsArray = [].slice.call(HTMLCollection);
var elementsArray = Array.from(HTMLCollection); [ES6]
```

Como convertir una HTMLCollection en un Array de JS para utilizar los métodos de los Arrays

```
var arr = [];
[].push.apply(arr, HTMLCollection);
```

Un poco más eficiente

Objetos del tipo Node

Propiedades

childNodes
firstChild
lastChild
nextSibing
nodeName
nodeType
nodeValue
parentNode
previousSibling

Métodos

appendChild()
cloneNode()
compareDocumentPosition()
contains()
hasChildrenNodes()
insertBefore()
isEqualNode()
removeChild()
replaceChild()

Métodos *document*

document.createElement()
document.createTextNode()

Propiedades *Element*

innerHTML
outerHMTL
textContent
innerText
outerText
firstElementChild
lastElementChild
previousElementChild
children

Métodos *Element*

insertAdjacentHTML()

Nodos. Propiedades (algunas)

- `id`: indica el identificador del nodo.
- `nodeType`: que indica el tipo de nodo que es por medio de un número que indica el tipo.
- `nodeName`: que devuelve el tipo de etiqueta HTML que representa al nodo. Hay navegadores que lo devuelven en mayúsculas y otros en minúsculas.
- `className`: que devuelve el nombre de la clase que haya sido asignado al nodo.
- `nodeValue`: que almacena el texto contenido en los nodos de tipo texto, es `null` para los nodo de tipo elemento.
- `innerHTML`: cadena que contiene el código HTML asociado con el nodo sin las etiquetas HTML.
- `outerHTML`: igual que `innerHTML` pero con las etiquetas HTML.

3. LOCALIZANDO ELEMENTOS 

Por el atributo `id`

Por el atributo `name`

Por el tipo de etiqueta

Por la clase o clases CSS

Por coincidencia de selector CSS

Encontrando elementos

Por el atributo `id`

Por id

Obtenemos un elemento con el `id` dado o `null` si no se encuentra.

```
var nodo = document.getElementById('nombre_id');
```

Es la forma más sencilla y más utilizada para seleccionar elementos.

Los tipos devueltos en las diapositivas que siguen se han especificado de acuerdo con los navegadores Chrome y Firefox

Encontrando elementos

Por el atributo **name**

Por nombre - *document-*

Obtenemos una `NodeList` con los elementos que tengan el atributo *nombre* del argumento. Si no se encuentran, la longitud de la `NodeList` será `0`.

```
var nodos = document.getElementsByName('nombre');
```

El atributo *nombre* no tiene porque ser único pero solo es válido en unos pocos elementos: elementos de formulario, iframes e img.

Dar nombre a `<form>`, `<img>`, `<iframe>`, etc hace que se creen propiedades con ese nombre con el valor del atributo en `document` (siempre, por supuesto, que ya no cuente con esa propiedad).

```
<form name='uno'>
var nodoF = document.uno;
```

Encontrando elementos

Por el tipo de etiqueta

Por etiqueta -*document* y *element*-

Obtenemos una `HTMLCollection` con los elementos del tipo de *etiqueta* pedido. Si no se encuentran, la longitud de la `HTMLCollection` será `0`.

```
var nodos = document.getElementsByTagName('etiqueta');
```

Como en el caso anterior, los elementos de la `HTMLCollection` aparecen en el orden que están en el documento.

Las etiquetas en HTML no son sensibles a mayúsculas por lo que tampoco lo es su búsqueda. Se pueden obtener todos mediante el asterisco.

El objeto de tipo *element* también cuenta con el método `getElementsByTagName()` seleccionando los elementos descendientes del elemento desde el que se invoca la función.

```
var elemento = document.getElementById('nombre_id');
var nodos = elemento.getElementsByTagName('etiqueta');
```

Encontrando elementos

Por la clase o clases CSS

Por clase -*document* y *element*-

Obtenemos una `HTMLCollection` con los elementos que tengan la clase (o clases separadas por un espacio) como atributo. Si no se encuentran, la longitud de la `HTMLCollection` será `0`.

```
var nodos = document.getElementsByClassName('nombre_clase');
```

Se puede invocar a nivel de documento o a nivel de elemento.

El argumento es una cadena con el nombre de la clase o clases. En este caso, los elementos deben contar con todas las clases especificadas (separadas por espacios) sin importar su orden.

```
var log = document.getElementsById('log');
var fatal = document.getElementsByClassName('fatal error');
```

Encuentra todos los descendientes del elemento con id=log y que tengan la clase `"error"` y `"fatal"`

Encontrando elementos

Por coincidencia de selector CSS

Por selector CSS *-document y element-*

Podemos pedir el *primer elemento* con un selector CSS dado:

```
var nodo = document.querySelector('selector CSS');
```

Si no encuentra ningún nodo, devuelve `null`.

```
var nodo = document.querySelector('#miId');
```

Encontrando elementos

Por coincidencia de selector CSS

Por selector CSS *-document y element-*

Podemos pedir el todos los elementos que tengan aplicado el selector CSS dado:

```
var nodos = document.querySelectorAll('selector CSS');
```

Obtenemos una `NodeList` con los elementos que cumplan la condición. Si no se encuentran, la longitud de la `NodeList` será `0`.

```
var nodos = document.querySelectorAll('.miClase');
var nodos = document.querySelectorAll('div > ul + li');
var nodos = document.querySelectorAll('p, img');
```

Elementos coincidentes

Por coincidencia de selector CSS

Por selector CSS *-element-*

Podemos preguntar si un elemento se puede seleccionar por un selector CSS dado. Devuelve `true` o `false`:

```
if(document.body.matchesSelectorAll('selector CSS')){//true;}
```

Este método nos permite comprobar fácilmente si un elemento dado sería devuelto por `querySelector` o `querySelectorAll`.

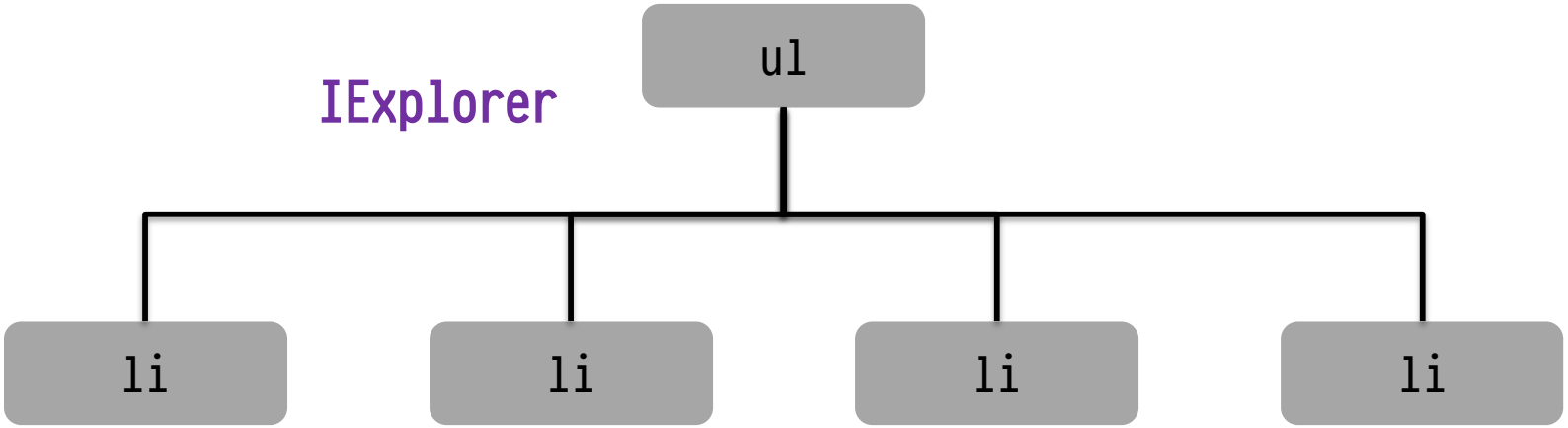
Encontrando elementos

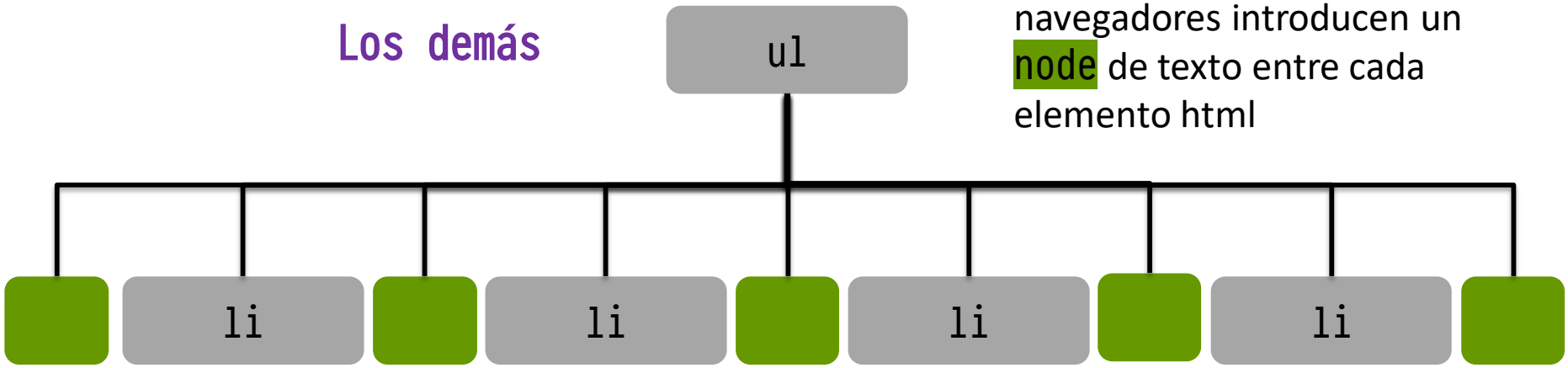
`document.all[]` (fuera de programa)

Antes de la estandarización del DOM IE4 introduzco la colección `document.all[]` donde están todos los elementos (salvo los nodos Text). Ahora está obsoleto pero aun así se sigue utilizando

```
document.all[0];           // El primer elemento del documento
document.all['navbar'];    // El elemento (o elementos) con nombre 'navbar'
document.all.navbar;       // Lo mismo
document.all.tags('div'); // Todos los div del docuemnto
document.all.tags('p')[0];  // El primer <p> del docuemnto
```

Inserción de node text



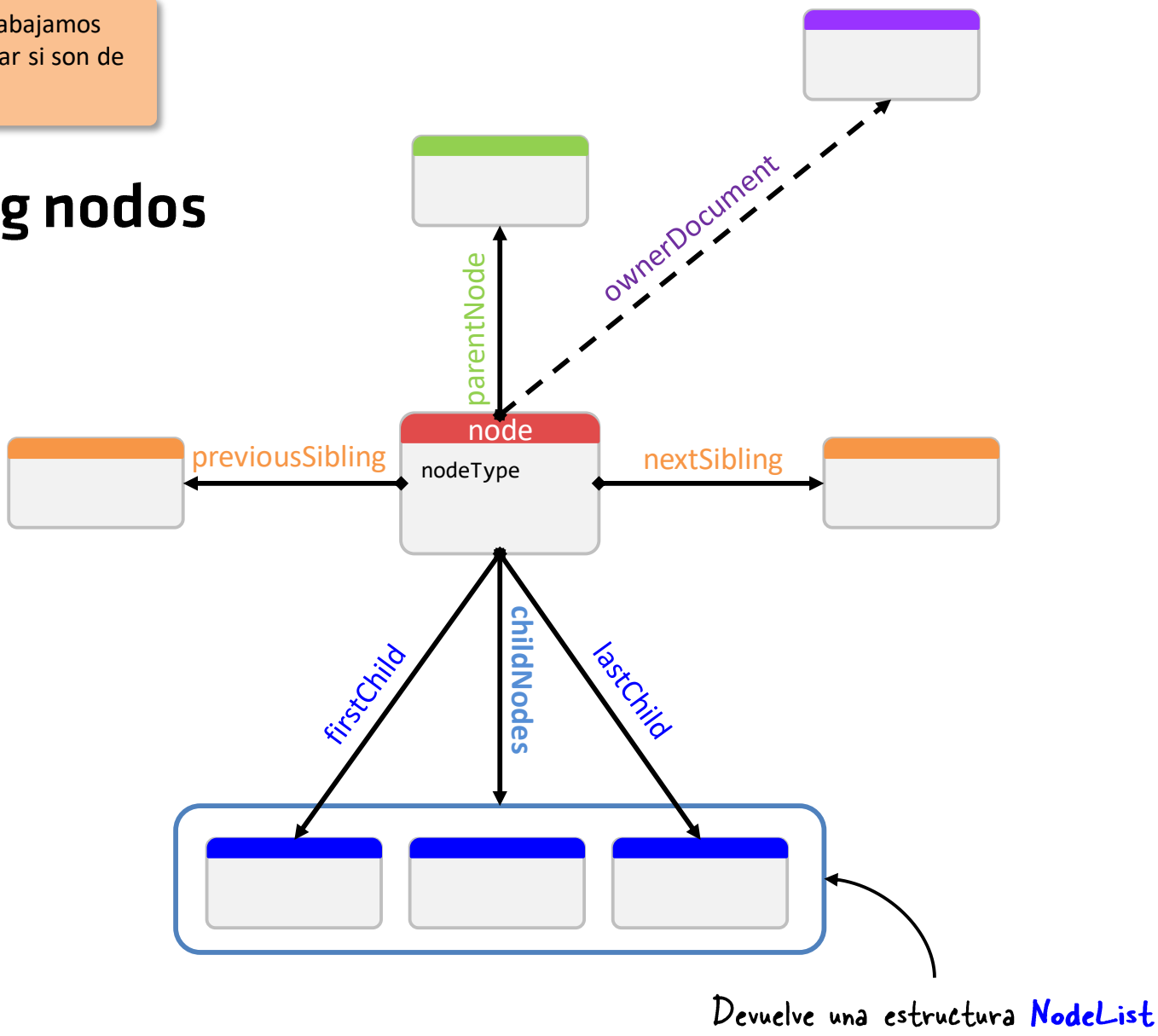


Salvo IExplorer, los demás navegadores introducen un node de texto entre cada elemento html

{cliente} daw.cierva@gmail.com

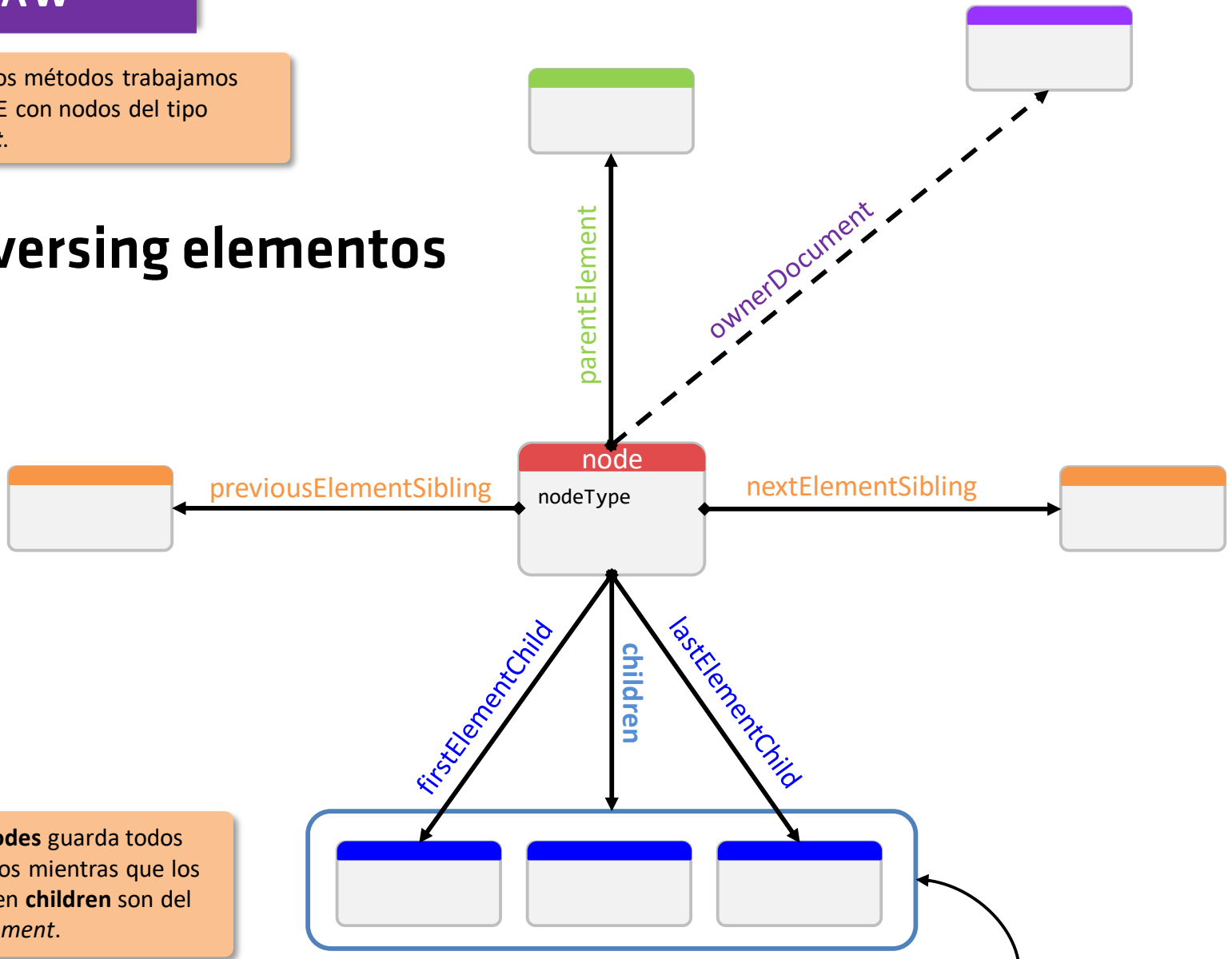
Con estos métodos trabajamos con nodos si especificar si son de *texto* o de *elementos*.

Traversing nodos



! Con estos métodos trabajamos SIEMPRE con nodos del tipo ***element***.

Traversing elementos



! **childNodes** guarda todos los nodos mientras que los nodos en **children** son del tipo *element*.

4. MODIFICANDO EL DOM

4.1 Cambiar la estructura

4.1.1 Agregar nodos

4.1.2 Quitar nodos

4.1.3 Contenido de un elemento

4.2 Modificando elementos

4.2.1 Atributos

4.2.2 Clases

4.2.3 Estilos

4.2.4 Eventos

Agregar, quitar y modificar elementos

Una vez que tenemos localizado el nodo o nodos podemos, básicamente, realizar dos acciones:

1. **Cambiar la estructura** del árbol DOM existente agregando, quitando o cambiando de sitio los nodos.
2. **Cambiar la información** asociada al nodo: estilos, eventos, atributos, etc.

4.1 CAMBIAR LA ESTRUCTURA

4.1.1 AGREGAR NODOS

Crear un nodo

Tipo *element*

Utilizamos el método `document.createElement` con argumento el nombre del tipo de etiqueta deseada.

```
var parrafo = document.createElement('p');
```

Tipo *text*

Utilizamos el método `document.createTextNode` con argumento el texto que queremos que guarde.

```
var nTexto = document.createTextNode('Tu texto aquí');
```

Una vez creado el nodo, se puede agregar al DOM

Clonar un nodo

Otra forma de crear un nodo es clonar uno ya existente. Cada nodo cuenta con el método `cloneNode` con un argumento que indica si se deben copiar todos sus descendientes (*true*) o solo él (*false*).

```
var log = document.getElementsById('log');
var logClone = log.cloneNode(false);
```

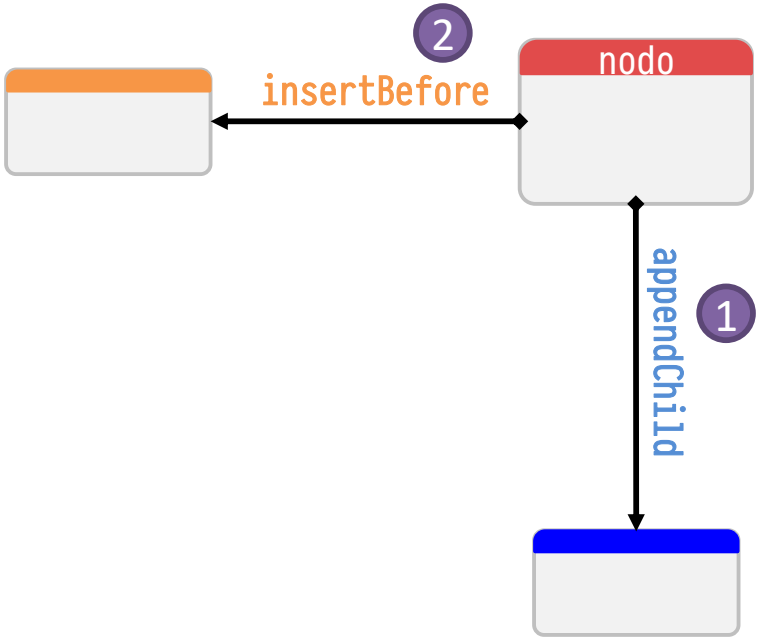
La clonación de un nodo copia todos sus atributos y valores, incluyendo los gestores de eventos intrínsecos (en la etiqueta HTML). **No copia** los gestores añadidos mediante *addEventListener* o los asignados a las propiedades de los elementos. (por ejemplo *node.onclick* = fn). Aún así, devuelve un objeto Node correspondiente no un NodeList

Por problemas de compatibilidad se recomienda poner siempre, con el valor adecuado, el argumento del método.

Agregar un nodo

Para agregar un nuevo nodo a DOM necesitamos hacerlo con relación a uno ya existente:

1. **Como hijo** de un nodo dado: `nodo.appendChild`
2. **Como hermano anterior** a un nodo dado: `nodo.padre.insertBefore`



En ambos casos, la agregación se realiza desde el padre

Agregar elemento: como hijo

1. Creamos el elemento y su contenido

```
var parrafo = document.createElement('p');
var nTexto = document.createTextNode('Tu texto aquí');
```

```
<div id='contenedor'></div>
```

Vamos a agregar los nodos creados a un div existente

2. Agregamos el nodo de texto al elemento párrafo

```
parrafo.appendChild(nTexto);
```

3. Agregamos el elemento párrafo al elemento div, que lo tenemos que localizar primero

```
nDiv = document.getElementById('contenedor');
nDiv.appendChild(parrafo);
```

```
<div id='contenedor'>
  <p>Tu texto aquí</p>
</div>
```

Resultado final

{cliente} daw.cierva@gmail.com

Agregar elemento: como hermano mayor

```
<div id='contenedor'>
  <p>Tu texto aquí</p>
</div>
```

Vamos a insertar otro párrafo en el div delante del que ya tenemos.

1. Creamos el elemento y su contenido

```
var parrafo = document.createElement('p');
var nTexto = document.createTextNode('Voy el primero');
```

2. Agregamos el nodo de texto al elemento párrafo

```
parrafo.appendChild(nTexto);
```

3. Agregamos el elemento párrafo al elemento div, que lo tenemos que localizar primero

```
hijo = document.getElementById('contenedor').firstElementChild;
hijo.parentElement.insertBefore(parrafo, hijo);
```

```
<div id='contenedor'>
  <p>Voy el primero</p>
  <p>Tu texto aquí</p>
</div>
```

Resultado final

DocumentFragment

La creación y uso de un nodo DocumentFragment proporciona un documento ligero DOM, residente en la memoria, que es externo al árbol de DOM activo. Sus nodos pueden manipularse fácilmente en la memoria y luego se anexarlo al DOM.

```
var trozoDOM = document.createDocumentFragment();
```

- Un fragmento de documento puede contener cualquier tipo de nodo (excepto <body> o <html>)
- El DocumentFragment en sí no se agrega al DOM cuando se agrega un fragmento. El contenido de él, sí.
- Cuando se anexa un fragmento de documento al DOM, éste se transfiere del DocumentFragment al lugar en el que se anexa. Desaparece de la memoria.

DocumentFragment

Al nodo DocumentFragment se le agregan elementos con los métodos vistos hasta ahora: insertBefore y appendChild; él mismo se agrega al DOM con estos mismos métodos. También se puede utilizar innerHTML (ver más adelante). Al ser un nodo, también se puede clonar.

```
var ulElm = document.querySelector('ul');
var docFrag = document.createDocumentFragment();

// agregar los li al fragmento
["blue", "green", "red", "blue", "pink"].forEach(function(e) {
  var li = document.createElement("li");
  li.appendChild(document.createTextNode(e))
  li.style.color = e;
  docFrag.appendChild(li);
});

// agregar el DocumentFragment al DOM
ulElm.appendChild(docFrag);
```

4.1.2 QUITAR NODOS

Quitar nodos

Quitar un nodo del DOM es parecido a agregarlo. El método que se utiliza es `removeChild` desde el padre por lo que es necesario *conocer* a su padre.

```
<div id='contenedor'>
  <p>Voy el primero</p>
  <p>Tu texto aquí</p>
</div>
```

Vamos a eliminar el primer párrafo del div.

1. Localizamos el nodo/elemento a eliminar

```
hijo = document.getElementById('contenedor').firstElementChild;
```

2. Lo eliminamos

```
hijo.parentElement.removeChild(hijo);
```

```
<div id='contenedor'>
  <p>Tu texto aquí</p>
</div>
```

Resultado final

Reemplazar nodos

Si lo que queremos es poner un nodo donde ya se encuentra otro, podemos utilizar `replaceChild` desde el padre.

```
<div id='contenedor'>
  <p>Tu texto aquí</p>
</div>
```

Vamos a reemplazar el párrafo del div por otro

1. Creamos el nodo/elemento que se va a intercambiar con el existente

```
nNuevo = document.createElement('section');
```

2. Localizamos el existente

```
nViejo = document.getElementById('contenedor').firstElementChild;
```

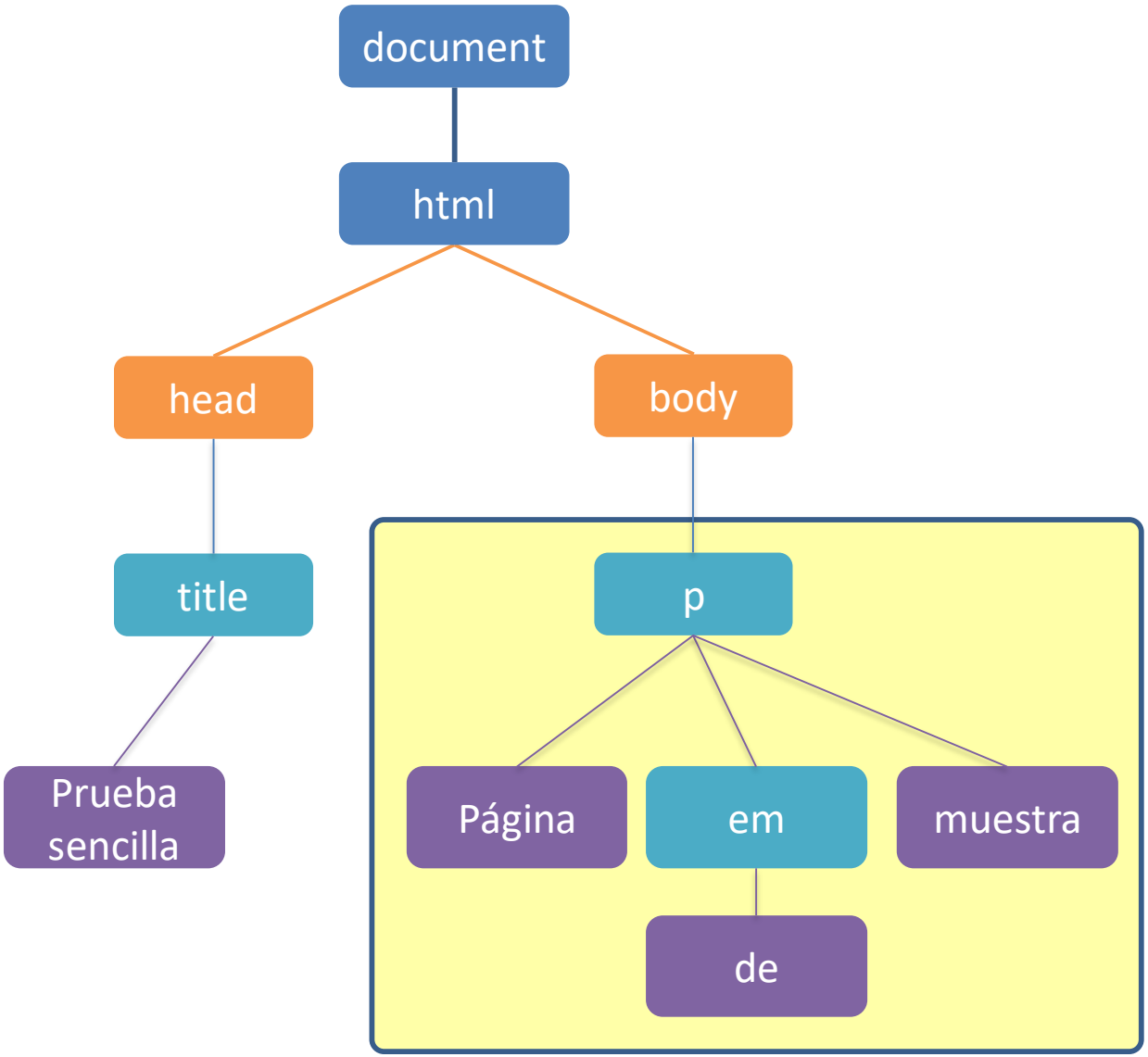
3. Lo reemplazamos

```
nViejo.parentElement.replaceChild(nNuevo, nViejo);
```

```
<div id='contenedor'>
  <section></section>
</div>
```

Resultado final

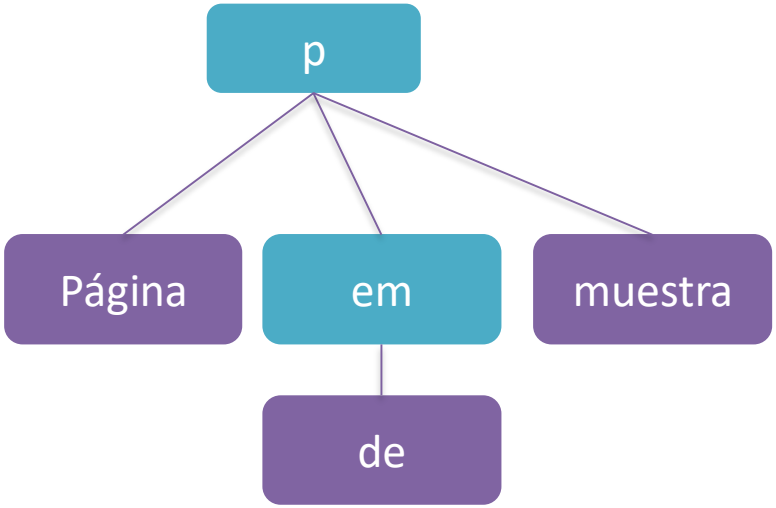
4.1.3 CONTENIDO DE UN ELEMENTO
(Como agregar/quitar nodos de otras formas)



¿Cuál es el contenido de <p>?

Hay 3 formas de contestar a la pregunta:

1. El contenido es la cadena HTML "Página `<em>de</em>` muestra"
2. El contenido es la cadena "Página de muestra"
3. El contenido es un nodo `Text`, un nodo `Element` que a su vez tiene un nodo hijo `Text` y otro nodo `Text` (la forma vista hasta ahora).



{cliente} daw.cierva@gmail.com

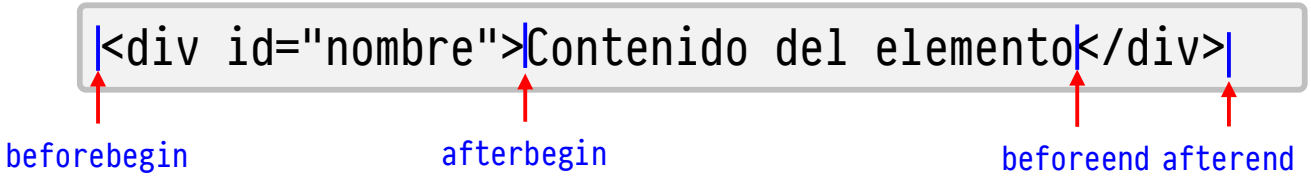
1. El contenido es la cadena HTML

Utilizaremos la propiedad `innerHTML` con la que se puede asignar o leer una cadena de texto donde el navegador interpretará las posibles etiquetas HTML.

```
nDiv = document.getElementById('contenedor');
nDiv.innerHTML = '<p>Voy el primero</p><p>Tu texto aquí</p>';
```

También contamos con la propiedad `outerHTML` que hace lo mismo que `innerHTML` pero incluyendo la etiquetas de inicio y cierre del elemento sobre el que estamos trabajando.

También tenemos el método `insertAdjacentHTML(posición, cadena)` que inserta el argumento *cadena* donde se indique con el argumento *posición*. Ésta puede ser `beforebegin`, `afterbegin`, `beforeend` o `afterend`



2. Como texto puro

En los nodos tipo `HTMLElement`, utilizaremos la propiedad `textContent` con la que se puede asignar o leer una cadena de texto donde el navegador **no** interpretara las posibles etiquetas HTML.

```
var parrafo = document.createElement('p');

parrafo.innerText = 'Voy el primero';
parrafo.textContent = 'Voy el primero';
```

IE

W3C, Opera, Safari, Firefox

En los nodos tipo `Text`, utilizaremos la propiedad `nodeValue` con la que se puede asignar o leer una cadena de texto donde el navegador **no** interpretara las posibles etiquetas HTML.

4.2 MODIFICANDO ELEMENTOS

Modificar un elemento

Las 3 acciones que se pueden realizar sobre el contenido de un elemento son:

1. **Cambiar sus atributos.**
2. **Cambiar el estilo CSS aplicado.**
3. **Agregar o quitar un gestor de eventos.**

4.2.1 ATRIBUTOS

Los atributos HTML como propiedades de elementos

Los objetos `HTMLElement` que representan los elementos HTML de un documento definen propiedades de escritura/lectura que reflejan los atributos de los elementos.

```
var f = document.forms[0];
f.action = "http://www.allavoy.com/enviar.php";
f.method = "POST";

var imagen = documento.getElementById("miImagen");
var urlImagen = imagen.src;
```

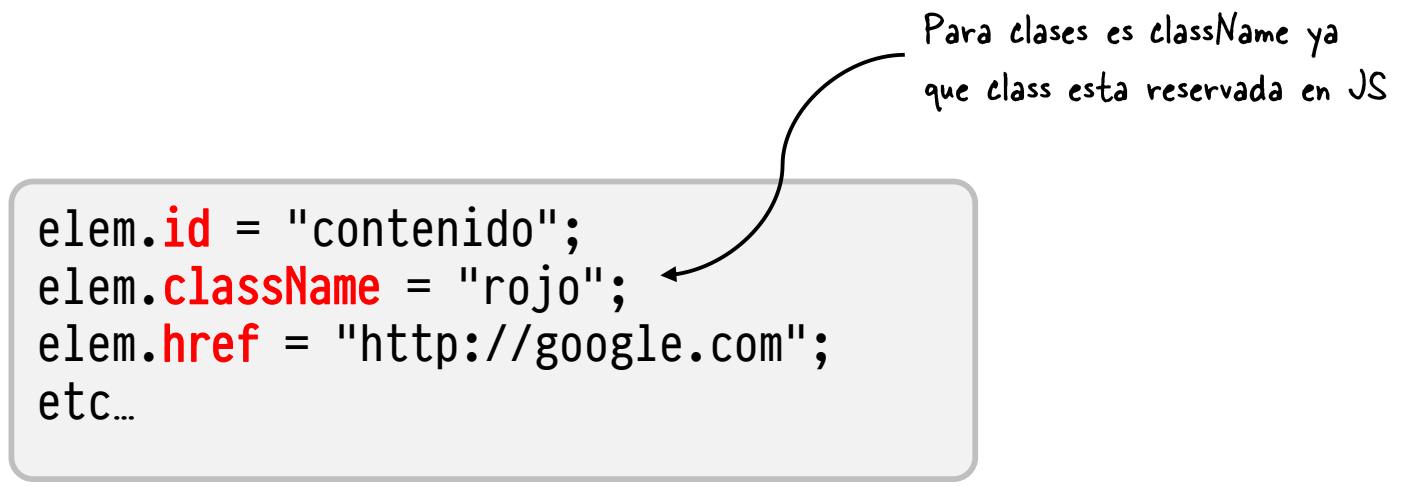
Los atributos HTML no son sensible a mayúsculas pero si lo son los nombres correspondientes en JS. Para convertir el nombre de un atributo a una propiedad JS, se escribe en minúsculas. Si el atributo tiene más de una palabra, la primera letra de cada una de ellas, después de la primera, se pone en mayúsculas (notación *camel*).

Los atributos HTML como propiedades de elementos

Algunos atributos son palabras reservadas en JS. En ese caso, se prefija el nombre con `html`. El caso de `class` es especial. La propiedad recibe el nombre de `className`

```
Atributo for de <label> → htmlFor
```

Las propiedades que representan atributos HTML representan un valor de cadena normalmente. Cuando el atributo es *boolean* o *numérico*, los valores de las propiedades son *booleanas* o *numéricas*. Los atributos de gestores de eventos son todos objetos `Function` (o `null`)



{cliente} daw.cierva@gmail.com

Atributos dataset

HTML5 proporciona una solución para agregar atributos no estándar pero de forma que el documento HTML siga siendo válido: cualquier atributo que empiece por el prefijo `data-` se considerará valido.

```
<div data-nombre="juan" data-apellidos="Pérez Pérez"></div>
```

HTML5 también define la propiedad `dataset` en los objetos *Element*. Esta propiedad referencia un objeto cuyas propiedades se corresponden con los atributos `data-` pero sin el prefijo.

```
data-nombre → dataset.nombre
data-apellidos → dataset.apellidos

data-x → dataset.x
data-otra-x → dataset.otraX
data-y-otra-mas → dataset.yOtraMas
```

Atributos como nodos Attr

Hay otra forma de trabajar los atributos de un elemento: el tipo Node define la propiedad attributes que es null en cualquier objeto que no sea HTMLElement. Para estos últimos, attributes es un objeto como un array de solo lectura que representa a todos los atributos de un elemento por medio de objetos Attr. (También se puede obtener mediante el método getAttributeNode())

```
<div id="id1" name="uno" dia="5">…</div>
```

```
var div = document.getElementById('id1');
```

> ! Esto ha cambiado con el DOM4: Attr ya no dependerá de Node. Se recomienda no utilizarlo hasta que no salga el estándar definitivo

```
div.attributes;                 // {0:id, 1:name, 2:dia} → array objetos Attr
div.attributes[0];              // id = "id1"
div.attributes['dia'];          // dia = "5"
div.getAttributeNode('dia')     // dia = "5"
```

Leyendo/escribiendo atributos no HTML

Para esta función contamos con los métodos `getAttribute()` y `setAttribute()`, que son universales.

```
var imagen = document.images[0];
var ancho = parseInt(image.getAttribute("WIDTH"));
imagen.setAttribute("class", "thumbnail");
```

En este caso, los atributos se tratan como cadenas (a la hora de escribir y devolver un valor) y se utilizan nombres de atributos estándar incluso cuando están reservados por JS. Los atributos no son sensibles a mayúsculas.

El resultado de utilizar estos métodos puede ser un documento HTML no validable.

También contamos con los métodos `hasAttribute()` y `removeAttribute()` que comprueban la existencia de un atributo y que lo eliminan totalmente, respectivamente.

4.2.2 CLASES

class

HTML5 ha introducido una serie de cambios para hacer el trabajo con clases CSS más fácil.

```
getElementsByClassName()
className
classList
```

La propiedad `classList` es una instancia de un nuevo tipo de colección llamada DOMTokenList. Tiene la propiedad *length* para indicar cuantos elementos contiene. Los elementos se puede obtener mediante el método `item(índice)` o utilizando la notación de paréntesis cuadrados. También cuenta con los siguiente métodos:

```
add(valor) → Agrega una cadena a la lista. Si existe, no se agrega.
contains(valor) → Indica si existe una cadena en la lista (true) o no (false).
remove(valor) → Borra una cadena de la lista.
toggle(valor) → Si el valor existe en la lista, lo quita. Si no, lo agrega.
```

```
forEach(función) → Igual que el método de array.
```

En algunas implementaciones

class

```
// Recorrer la lista de clases de un elemento
for (var i=0, len=div.classList.length; i<len; i++) {
  hacerAlgo(div.classList[i]);
}

// O, si está implementado
div.classList.forEach(clase => hacerAlgo(clase));
```

```
// eliminar la clase 'disabled'
div.classList.remove("disabled");

// agregar la clase 'actual'
div.classList.add("actual");

// eliminar/agregar la clase 'usuario'
div.classList.toggle("usuario");

// ¿la lista tiene alguna clase en concreto?
if(div.classList.contains("bd") &&
         !div.classList.contains("disabled")) {}
```

4.2.3 ESTILOS

Modificar un elemento. Estilos

Cada atributo CSS posee una propiedad del DOM equivalente en el objeto `style` del correspondiente elemento, formándose con el mismo nombre del atributo CSS pero sin los guiones, convirtiendo la primera letra de las palabras que van después de un guion, a mayúscula.

```
un-atributo-css → unAtributoCss
border-top-color → borderTopColor
```

La forma de acceder al estilo de un elementó será:

```
nodo_elem.style.propiedadCss
```

Se puede trabajar también con las propiedades 'atajo'

```
nodo_elem.style.border="1px solid blue";
```

Modificar un elemento. Estilos

```
<div id='contenedor'></div>
```

Vamos a poner un borde de 2 puntos, rojo, y el fondo verde

1. Localizamos el div

```
elem = document.getElementById('contenedor');
```

2. Aplicamos los estilos

```
elem.style.border = '2px solid red';
elem.style.backgroungColor = 'green';
```

Resultado final

```
<div id='contenedor'
  style="border: 2px solid red; background-color: green;">
</div>
```

! Con lo que se trabaja con los estilos en DOM es con el estilo empotrado (en la etiqueta html) del elemento. Tanto si lo escribimos como si lo leemos.

Reglas CSS: CSStyleRule

Cada una de los ficheros CSS o etiquetas style se encuentran en `document.styleSheets` que es un array de `CSSStyleSheet`. Dentro, cada una contiene un array, `cssRules`, que tiene una `CSStyleRule` por cada regla que aparece en el fichero.

Nombre	Descripción
cssText	Devuelve el texto de una regla.
parentRule	Si la regla es importada, esta es la regla. Si no es null
parentStyleSheet	La hoja de estilos de la que forma parte la regla
selectorText	Devuelve el texto de la regla.
style	Un objeto CSSStyleDeclaration que permite escribir y leer valores de estilo de la regla
type	Una constante que indica el tipo de regla. Para las reglas de estilo es siempre 1

Métodos de `CSSStyleSheet` :

Nombre	Descripción
insertRule(regla[, posición])	Agregar una regla.
deleteRule(posición)	Borra la regla que ocupa la posición dada.

Reglas CSS

```
div.caja {
  backgound-color: blue;
  width: 100px;
  height: 200px; }
```

Asumimos que esta es la primera y única regla de la hoja de estilos

```
var hoja=document.styleSheets[0];
var reglas=hoja.cssRules || hoja.rules;        // la lista de reglas
var regla=reglas[0];
alert(regla.selectorText);                     // 'div.caja'
alert(regla.style.cssText);                    // todo el código css
alert(regla.style.backgroundColor);            // 'blue'
alert(regla.style.width);                      // '100px'
alert(regla.style.height);                     // '200px'
regla.style.backgroundColor = "red";           // a rojo
```

```
hoja.insertRule("body {background-color: silver}", 0);
hoja.deleteRule(0);
```

4.2.4 EVENTOS

Modificar un elemento. Eventos

evento

Cosas que les suceden a los elementos en DOM

- Se carga (`load`) la página
- Se pulsa (**clicked**) en un elemento
- Se pasa el ratón por encima **(mouseover)** de un elemento
- etc.

Cuando sucede algo así seguramente queramos hacer algo. Por ejemplo, ejecutar un *script* cuando se cargue una página. O realizar una acción cuando se pulse un elemento

Básicamente nos encontraremos con 2 tipos de eventos: los que genera el proprio navegador (**load, submit**, …) y los que genera el usuario (**mouseover, keypress, click,** …)

Eventos. Agregar un manejador

Manejador de eventos

Función que se ejecutará cuando se produzca el evento.

```
<div id='contenedor'>
   ¡Púlsame!
</div>
```

Vamos a agregar un manejador de evento para cuando el usuario pulse en él

Una vez localizado el elemento, agregamos el manejador:

La función que se ejecutará cuando se dispare el evento

```
elem.addEventListener('click', pulsado, false);
```

El evento controlado

false = bubble
true = capture


```
elem.attachEvent('onclick', pulsado);
```

{cliente} daw.cierva@gmail.com

Eventos. Quitar un manejador

Una vez localizado el elemento, le quitamos el manejador:

La función que se ejecutará cuando se dispare el evento

```
elem.removeEventListener('click', pulsado, false);
```

El evento controlado

false = bubble
true = capture

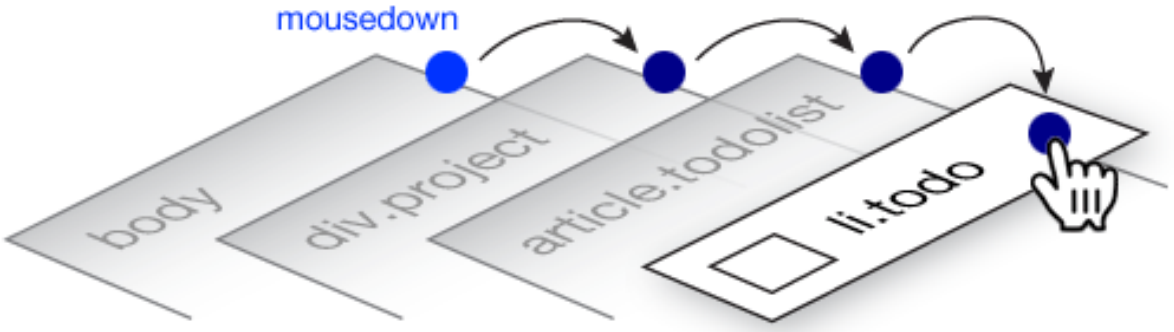
```
elem.detachEvent('onclick', pulsado);
```

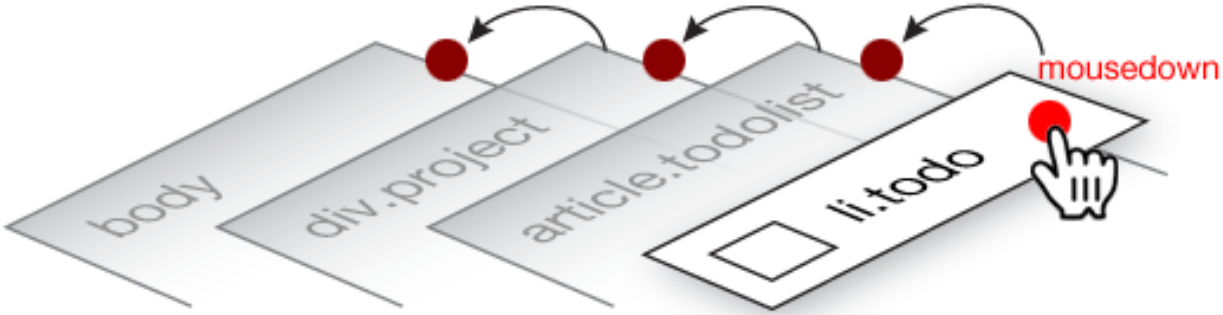
Función universal

```
function addEvent(elemento, tipo, fn) {
  if (elemento.addEventListener) {
    elemento.addEventListener(tipo, fn, false);
  } else {
    elemento.attachEvent("on" + tipo, fn);
  }
}
```

capturing/bubbling

Primera fase





Segunda fase